Project Design Report

Ian Waldschmidt [isw5@case.edu](mailto:isw5@case.edu) EECS 341 - Gurkan Bebek

Justin Lee [jpl88@case.edu](mailto:jpl88@case.edu) Introduction to Database Systems

Jacob Alspaw [jaa134@case.edu](mailto:jaa134@case.edu)

**1. Derive a global relational database schema and normalize to at least 3NF. Consider and discuss all functional dependencies that hold on your schemas.**

Our global relational database schema has 3 kinds of functional dependencies. We will discuss each one in more detail below:

* FD’s in the form “primary\_key → attribute”
  + This is where the vast majority of our FD’s are (“username → password,” for example). However, we can say that each of these FD’s follow the rules of 3NF (and also BCNF) without going into the specifics of each FD. For each of these FD’s, the table they apply to only has 1 candidate key, and that is the primary key. Therefore, for each of these FD’s, “primary\_key” is a candidate key of the table in question, so these FD’s don’t violate 3NF.
* FD’s in the form “foreign\_key → attribute”
  + These occur with 1-to-many relationships, the only case being “mediaId → username.” While this seems different when represented in the ER diagram, in reality this is identical to the first case. This is because when these tables are represented in table form in a given DBMS, these 1-to-many relationships don’t exist in separate tables; they manifest themselves as a foreign key on the table on the “many” side of the relationship. For example, in the case of “mediaId → username” (“mediaId” being on the “Media” table and “username” being on the “User” table), the 1-to-many relationship between the 2 tables (many users can upload media, but each piece of media only has 1 “uploader”) manifests itself as a foreign key on the “Media” table. And now this kind of FD looks identical to the first case since “mediaId” is now the PK of the table and “username” is now just another attribute in the table. This means these kinds of FD’s preserve not only 3NF, but also BCNF.
* Other FD’s
  + There is only 1 FD in our entire global schema that doesn’t fit into this kind of mold; “email → username” on the “Users” table (since each user must register with a unique email address). Unfortunately, this doesn’t satisfy the BCNF requirements because it isn’t a trivial FD and the LHS of the FD isn’t a superkey. However, this satisfies the 3NF requirements because “username” is the PK of the “Users” table, meaning the RHS of the FD is part of “some” candidate key of their respective table.

Because of the facts listed above, our global relational database schema is NOT already in BCNF form, but it IS already in 3NF form, so we will not make any changes to our schema since we will only have minimal data redundancy.

**2. Describe any design decisions for the global schema. Identify and explain integrity constraints of the following types: NULL, key, and referential-integrity.**

Looking at the schema in the Project Specification Report (found in the zip folder this document is in), we can easily see what kinds of integrity constraints we need for each attribute in each table. All underlined fields are primary keys, so they must be unique and not NULL. All red fields are foreign keys, so they must have referential integrity. No null fields are accepted.

**3. For all your relation schemas create base relations with the right attribute domains and with all the integrity constraints. Include your CREATE TABLE statements.**

We moved the CREATE TABLE statements to their own executable SQL file to make running each query easier. Please see the attached SQL file. We confirmed the statements worked by executing the query in MySQL.

**4. Write SQL queries for each operation in your narrative.**

All video titles

SELECT Media.name

FROM Media, Video

WHERE Media.mediaId = Video.mediaId

All picture titles

SELECT Media.name

FROM Media, Picture

WHERE Media.mediaId = Picture.mediaId

All music titles

SELECT Media.name

FROM Media, Music

WHERE Media.mediaId = Music.mediaId

All actors in a movie

SELECT Actor.name

FROM Actor, In, Movie, Video, Media

WHERE Actor.name = In.name

And In.videoId = Movie.movieId

AND Movie.videoId = Video.videoId

AND Video.mediaId = Media.mediaId

AND Media.name = ‘<title>’

All movies by a certain actor

SELECT Media.name

FROM Media, Video, Movie, In, Actor

WHERE Media.mediaID = Video.mediaId

AND Video.videoId = Movie.videoId

AND Movie.movieId = In.movieId

AND In.name = Actor.name

AND Actor.name = ‘<name>’

Users who accessed media today

SELECT User.username

FROM User, History

WHERE User.username = History.username

AND DATEDIFF(day, History.accessed, GETDATE()) = 0

All music a user has listened to

SELECT Media.name

FROM Music, Media, History, User

WHERE Music.mediaId = Media.mediaId

AND Media.mediaId = History.mediaId

AND History.username = User.username

AND User.username = ‘<username>’

All videos user has watched

SELECT Media.name

FROM Video, Media, History, User

WHERE Video.mediaId = Media.mediaId

AND Media.mediaId = History.mediaId

AND History.username = User.username

AND User.username = ‘<username>’

All pictures user has seen

SELECT Media.name

FROM Picture, Media, History, User

WHERE Picture.mediaId = Media.mediaId

AND Media.mediaId = History.mediaId

AND History.username = User.username

AND User.username = ‘<username>’

The name of the user’s most watched actor

SELECT ActorPopularity.name AS mostWatchedActor

FROM (

SELECT In.name, COUNT(In.name) as views

In, Movie, Video, History, User

WHERE In.movieId = Movie.movieId

AND Movie.videoId = Video.videoId

AND Video.mediaId = History.mediaId

AND History.username = User.username

AND User.username = ‘<username>’

GROUP BY In.name) AS ActorPopularity

WHERE views = MAX(views)

Searching for media by title

SELECT Media.name

FROM Media

WHERE Media.name LIKE ‘%<title>%’

Searching for movies by actor

SELECT Media.name

FROM Media, Video, Movie, In

WHERE Media.mediaId = Video.mediaId

AND Video.videoId = Movie.videoId

AND Movie.movieId = In.MovieId

AND In.name LIKE ‘%<name>%’

Searching for music by artist

SELECT Media.name

FROM Media, Music

WHERE Media.mediaId = Music.mediaId

AND Music.artist LIKE ‘%<name>%’

Media’s number of views

SELECT Media.name, COUNT(History.mediaID) AS viewCount

FROM Media, History

WHERE Media.mediaId = History.mediaId

Sort media by popularity

SELECT Media.name, COUNT(History.mediaID) AS viewCount

FROM Media, History

WHERE Media.mediaId = History.mediaId

ORDER BY viewCount DESC

Sort by title

SELECT Media.name

From Media

ORDER BY Media.name ASC

Sort media by rating

SELECT Media.name, AVG(Rated.rating) AS avgRating

FROM Media, Rated

WHERE Media.mediaId = Rated.mediaId

ORDER BY avgRating DESC

Sort videos by duration

SELECT Media.name, Video.duration

FROM Media, Video

WHERE Media.mediaId = Video.mediaId

ORDER BY Video.duration DESC

Get all media a user watched in the past week

Select Media.name

FROM Media, History, User

WHERE Media.mediaId = History.mediaId

AND History.username = User.username

AND User.username = ‘<username>’

AND DATEDIFF(day, History.accessed, GETDATE()) <= 7

Get media’s rating

SELECT AVG(Rated.rating) as avgRating

FROM Rated, Media

WHERE Rated.mediaId = Media.mediaId

AND Media.name = ‘<title>’

Get media’s comments

SELECT Comment.comment

FROM Comment, Media

WHERE Comment.mediaId = Media.mediaId

AND Media.name = ‘<title>’

Get a list of who uploaded each piece of media

SELECT Media.name, UploadedBy.username

FROM Media, UploadedBy

WHERE Media.mediaId = UploadedBy.mediaId

User’s most recent media access

SELECT Media.name, MAX(History.accessed) AS mostRecentAccessDate

FROM Media, History

WHERE Media.mediaId = History.mediaId

AND History.username = ‘<username>’

User’s first media access

SELECT Media.name, MIN(History.accessed) AS firstAccessDate

FROM Media, History

WHERE Media.mediaId = History.mediaId

AND History.username = ‘<username>’

Total amount of media

SELECT Count(Media.mediaId) AS amountOfMedia From Media

Total amount of videos

SELECT Count(Video.videoId) AS amountOfVideos From Video

Total amount of music

SELECT Count(Music.musicId) AS amountOfMusic From Music

Total amount of pictures

SELECT Count(Picture.pictureId) AS amountOfPictures From Picture

Function to check if password is valid for given user (returns 1 if true, 0 if false)

SELECT COUNT(\*)

FROM User

WHERE User.username = ‘<username>’

AND User.password = ‘<password>’

Check if username already exists (returns 1 if username valid, 0 if not)

SELECT (1 - COUNT(\*))

FROM User

WHERE User.username = ‘<username>’

All the actors in a movie

SELECT In.name

FROM In, Movie, Video, Media

WHERE In.movieId = Movie.movieId

AND Movie.videoId = Video.videoId

AND Video.mediaId = Media.mediaId

AND Media.name = ‘<title>’

The longest piece of music

SELECT Media.name, MAX(Music.duration) AS maxTime

FROM Media, Music

WHERE Media.mediaId = Music.musicId

The shortest piece of music

SELECT Media.name, MIN(Music.duration) AS minTime

FROM Media, Music

WHERE Media.mediaId = Music.musicId

The longest video

SELECT Media.name, MAX(Video.duration) AS maxTime

From Media, Video

WHERE Media.mediaId = Music.musicId

The shortest video

SELECT Media.name, MIN(Video.duration) AS minTime

From Media, Video

WHERE Media.mediaId = Music.musicId

Movies with a certain age rating

SELECT Media.name

FROM Media, Video

WHERE Media.mediaId = Video.mediaId

AND Video.videoId = Movie.movieId

AND Movie.ageRating = ‘<ageRating>’

User who has uploaded the most content

SELECT UploadCountByUser.username, MAX(UploadCountByUser.numUploads)

FROM (

SELECT UploadedBy.username, COUNT(UploadedBy.username) as numUploads

FROM UploadedBy) AS UploadCountByUser

**5. List of Project Members and a brief description of their roles.**

In general, we all worked together on the project. More specifically, Jacob worked on writing the SQL queries, Justin worked on defining integrity constraints and creating the tables, and Ian worked on improving the global schema and ensuring it was in 3NF. We all checked over each other’s work, and this document was made in Google Drive and worked on collaboratively.